UNIVERSITY OF EXETER
Q-STEP CENTRE

Q-Step

A step-change in quantitative social science skills

Funded by the Nuffield Foundation, ESRC and HEFCE

Graphical exploratory data analysis in Python

Lewys Brace
l.brace@Exeter.ac.uk

Exploratory data analysis (EDA)

Exploring your data is the a crucial step in data analysis. It involves:

- Organising the data set
- Plotting aspects of the data set
- Maybe producing some numerical summaries; central tendency and spread, etc.

"Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone."
- John Tukey.

Let’s get data

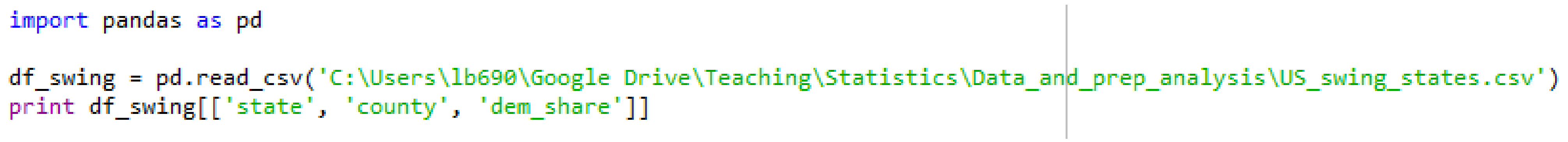
- Before we do anything, we’re going to need data to play with.
- Go to the following link, click “clone or download”, and then click “download ZIP”:

https://github.com/LewBrace/Python_for_EDA_workshop

- Extract the data sets to a folder that you’re going to remember the location of.

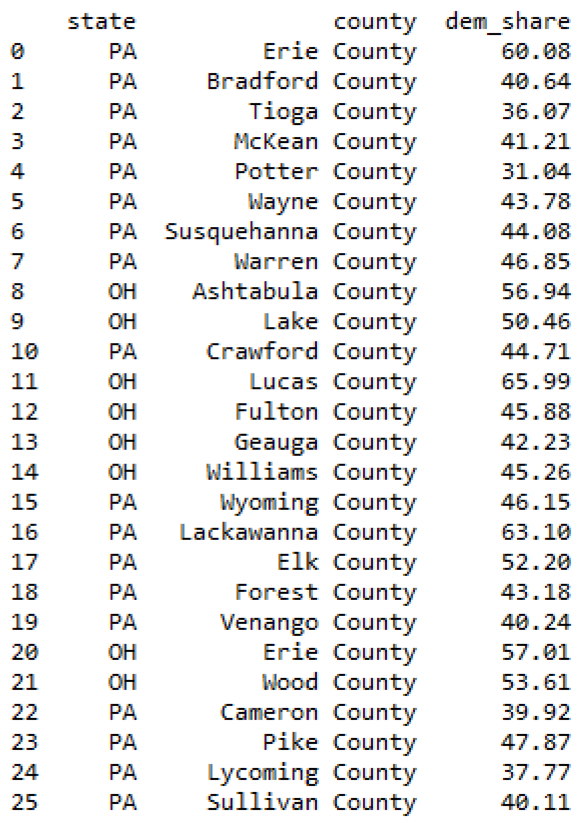
An example: 2008 US swing state election results

In:



```
import pandas as pd

df_swing = pd.read_csv('C:\Users\lb690\Google Drive\Teaching\Statistics\Data_and_prep_analysis\US_swing_states.csv')
print df_swing[['state', 'county', 'dem_share']]
```

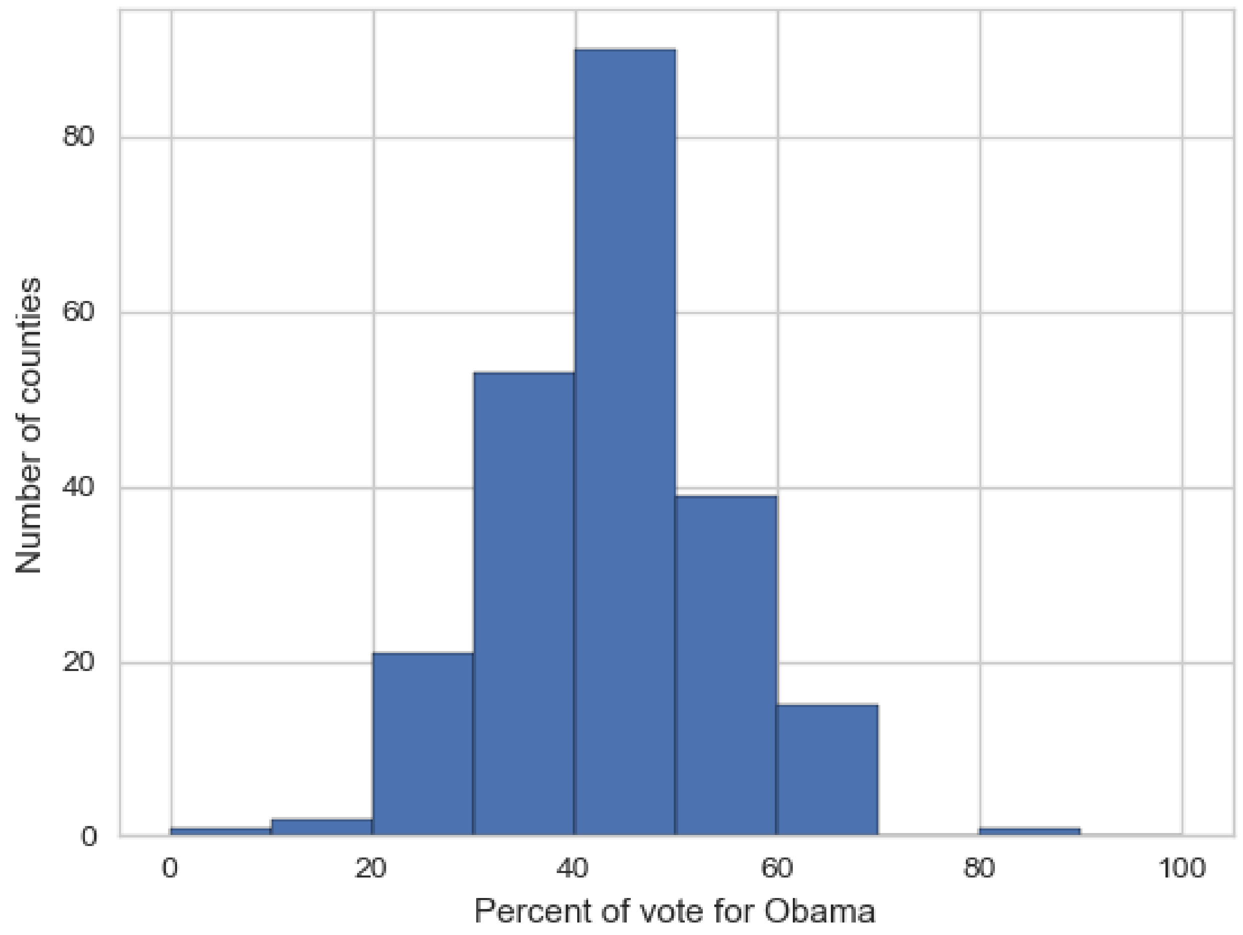
Out:



```
   state              county  dem_share
0     PA         Erie County      60.08
1     PA     Bradford County      40.64
2     PA        Tioga County      36.07
3     PA       McKean County      41.21
4     PA       Potter County      31.04
5     PA        Wayne County      43.78
6     PA  Susquehanna County      44.08
7     PA       Warren County      46.85
8     OH    Ashtabula County      56.94
9     OH         Lake County      50.46
10    PA     Crawford County      44.71
11    OH        Lucas County      65.99
12    OH       Fulton County      45.88
13    OH       Geauga County      42.23
14    OH     Williams County      45.26
15    PA      Wyoming County      46.15
16    PA   Lackawanna County      63.10
17    PA          Elk County      52.20
18    PA       Forest County      43.18
19    PA      Venango County      40.24
20    OH         Erie County      57.01
21    OH         Wood County      53.61
22    PA      Cameron County      39.92
23    PA         Pike County      47.87
24    PA     Lycoming County      37.77
25    PA     Sullivan County      40.11
```

- Here, we are only looking at the columns of immediate interest: the state, the country, and the share of votes that went the democratic Obama.
- We could spend time staring at these numbers, but that is unlikely to offer us any form understanding.
- We could begin by conducting all of our statistical tests.
- However, a good field commander does not go into battle without first doing a recognisance of the terrain…

- This is exactly what EDA is intended for.
- Graphical data analysis involves taking data presented in a tabular form and representing it graphically.
- As an example, lets take the democratic share of the vote in the counties of all three swing states and plot them as a histogram.
- The height of each of the bars is the number of counties that had a given level of support for Obama.
- Because there is more area in the histogram to the left of 50%, we can see that more people voted for John McCain than Obama.

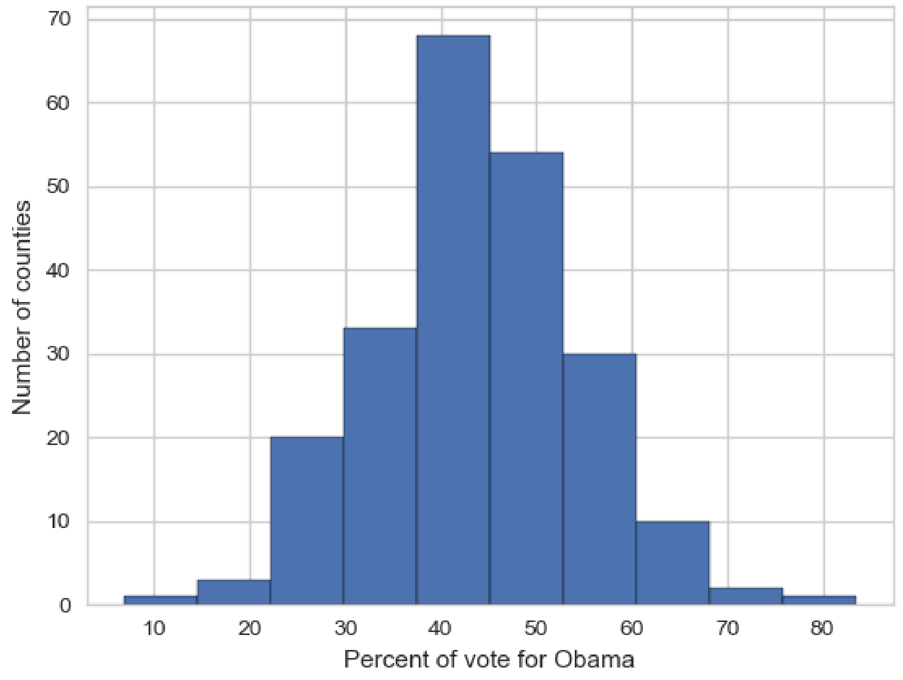


Plotting a histogram in Python

In:

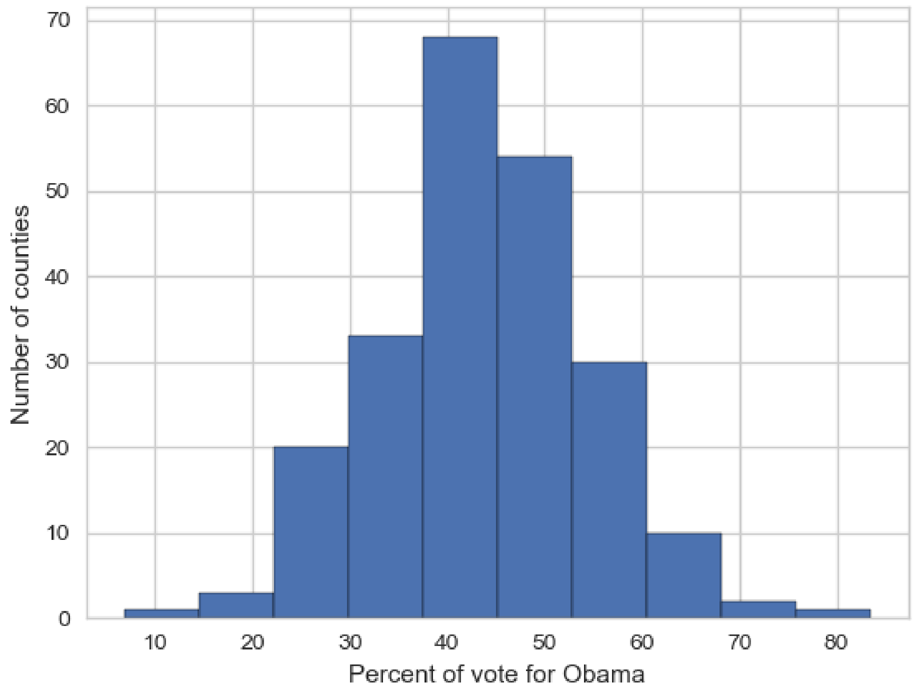
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df_swing = pd.read_csv('C:\Users\lb690\Google Drive\Teaching\Statistics\Data_and_prep_analysis\US_swing_states.csv')
_=plt.hist(df_swing['dem_share'], histtype='bar', ec='black')
_=plt.xlabel('Percent of vote for Obama')
_=plt.ylabel('Number of counties')

plt.show()
```

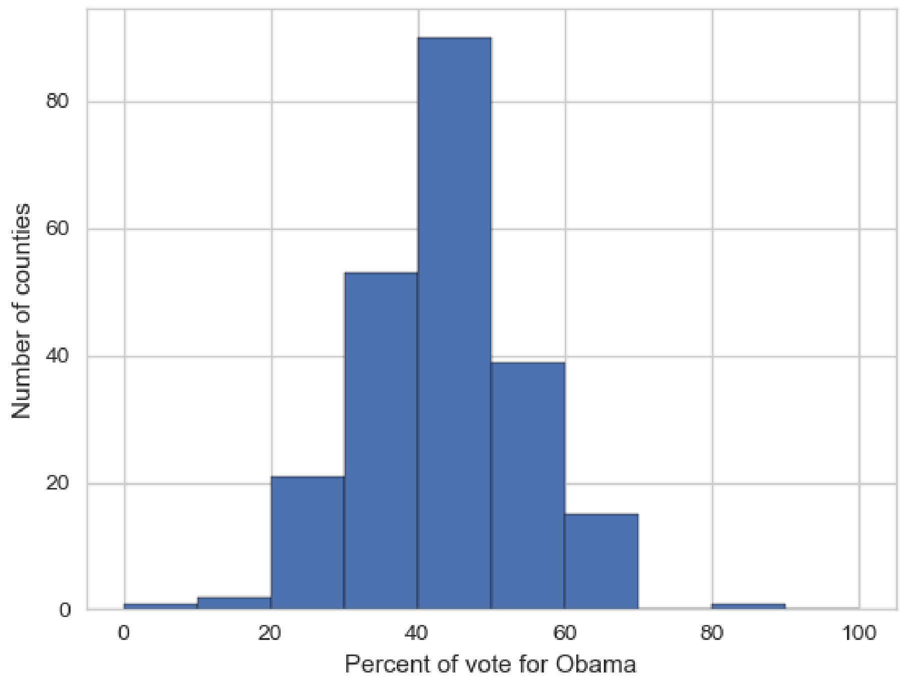
Out:



Bins

- You may have noticed the two graphs we've seen so far look different.
- This is because they have different binnings.
- The left graph used the default bins generated by plt.hist(), while the one on the right used bins that I specified.



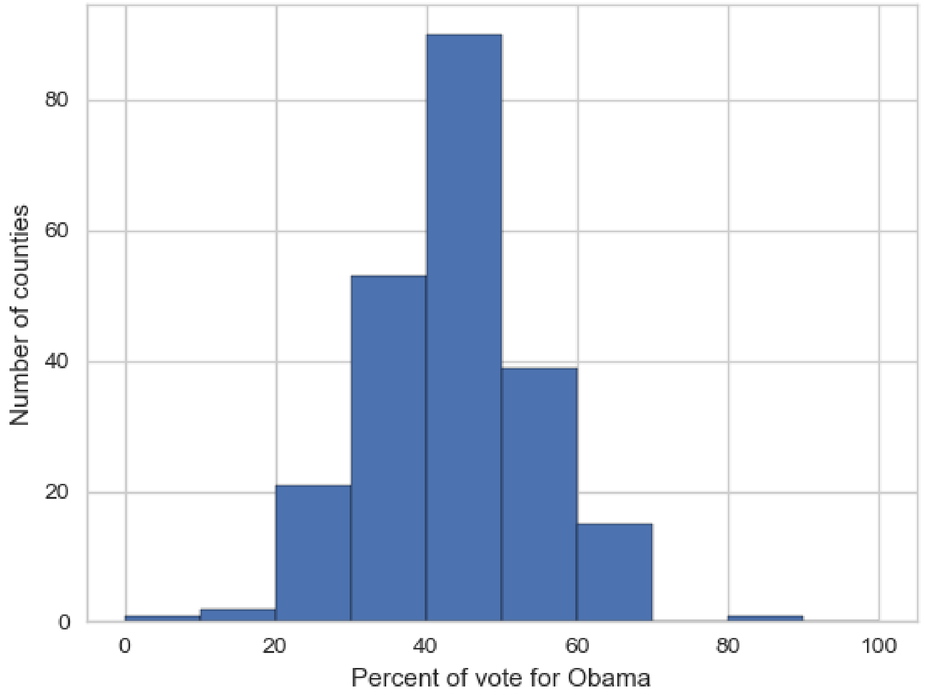


- I specified where the edges of the bars of the histogram are; the bin edges.

In:

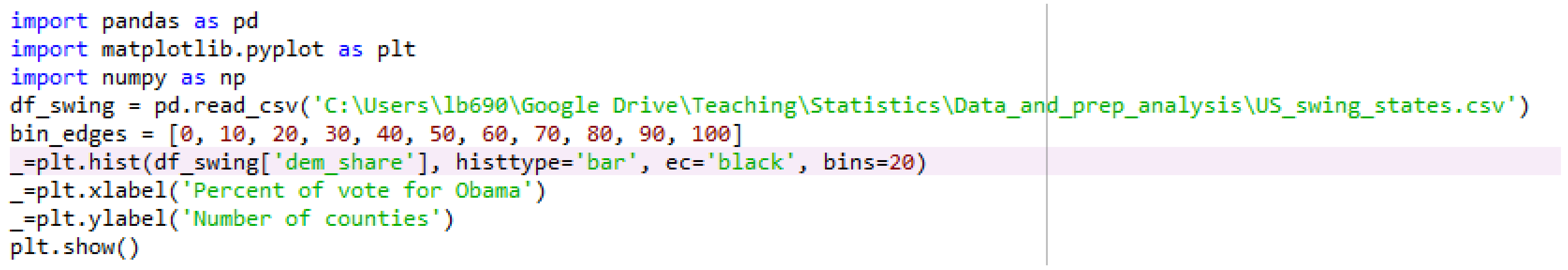
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df_swing = pd.read_csv('C:\Users\lb690\Google Drive\Teaching\Statistics\Data_and_prep_analysis\US_swing_states.csv')
bin_edges = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
_=plt.hist(df_swing['dem_share'], histtype='bar', ec='black', bins=bin_edges)
_=plt.xlabel('Percent of vote for Obama')
_=plt.ylabel('Number of counties')
plt.show()
```

Out:



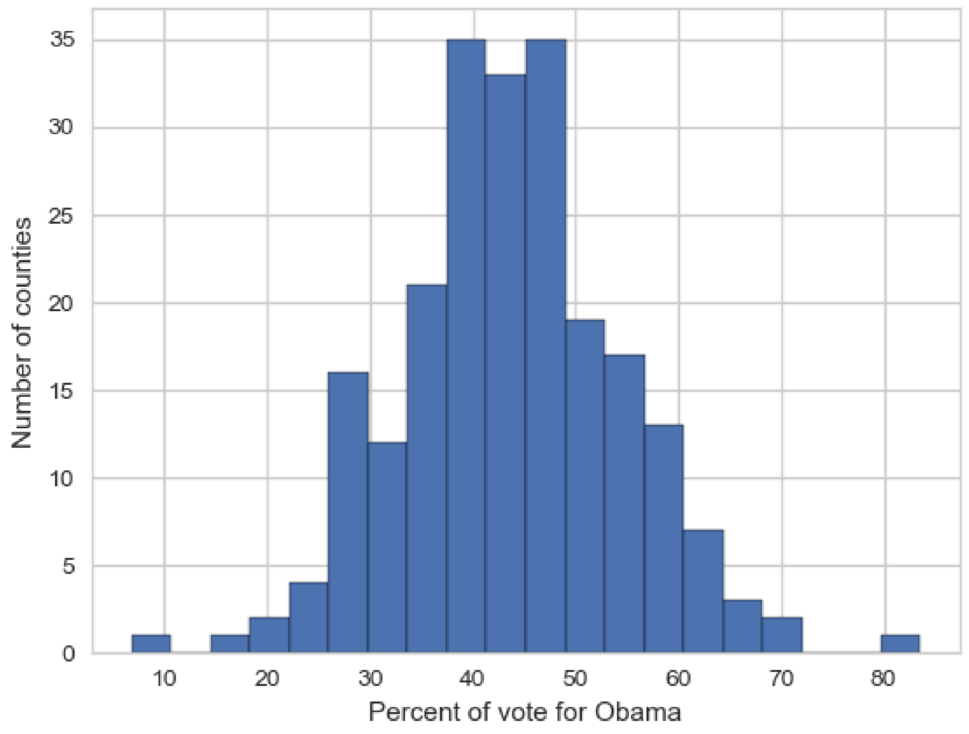
- You could specify the number of bins, and Matplotlib will automatically generated 20 evenly spaced bins.

In:



```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
df_swing = pd.read_csv('C:\Users\lb690\Google Drive\Teaching\Statistics\Data_and_prep_analysis\US_swing_states.csv')
bin_edges = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
_=plt.hist(df_swing['dem_share'], histtype='bar', ec='black', bins=20)
_=plt.xlabel('Percent of vote for Obama')
_=plt.ylabel('Number of counties')
plt.show()
```

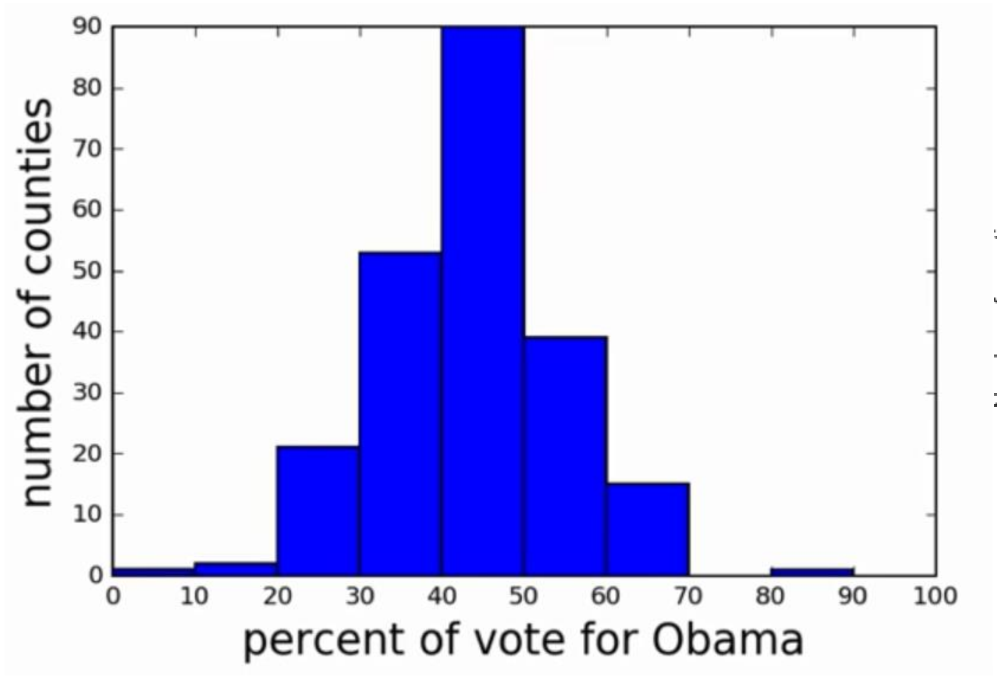
Out:

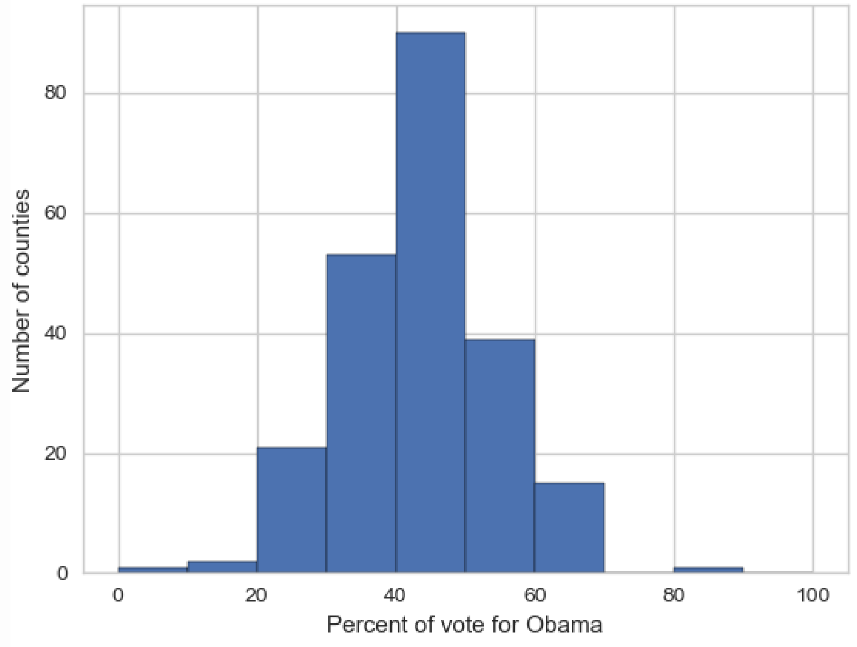


Note: plt.plot() returns three arrays that we are not interest in here, we only want the plot. We therefore assign a dummy variable of _ to them. This is common practice in Python.

Seaborn

- It’s still quite common for people to plot graphs using Matplotlib’s default style settings; i.e. left graph.
- Seaborn is a relatively new, Matplotlib-based statistical visualisation package; i.e. right graph.
- We set the script to use Seaborns default style settings with sns.set().





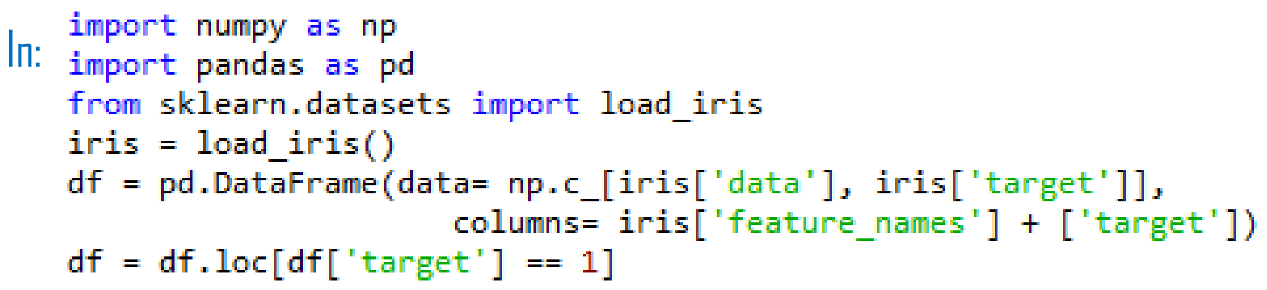
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
df_swing = pd.read_csv('C:\Users\lb690\Google Dr
sns.set()
_=plt.hist(df_swing['dem_share'])
_=plt.xlabel('Percent of vote for Obama')
_=plt.ylabel('Number of counties')
plt.show()
```

Practice: histogram

- For this exercises in this section, you will use a classic data set collected by botanist Edward Anderson and made famous by Ronald Fisher, one of the most prolific statisticians in history.
- Anderson carefully measured the anatomical properties of samples of three different species of iris, *Iris setosa*, *Iris versicolor*, and *Iris virginica*.
- You will be working with the measurements of petal length.

Import the data set

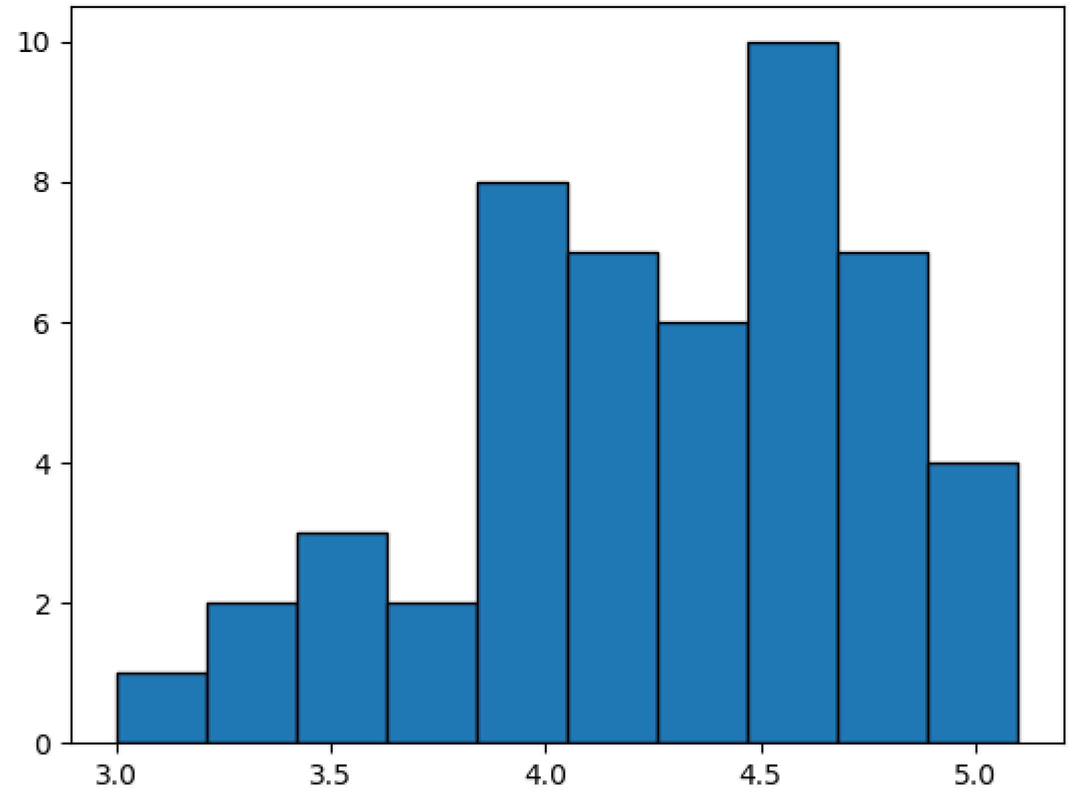
- This dataset comes as a default dataset in the scikit-learn package.
- We are first going to import the data set from this package and turn it into a Pandas data frame.
- We are then going to create a sub-dataframe, so that we have a dataframe that only contains the entries that are part of the *versicolor* family.



```
In: import numpy as np
    import pandas as pd
    from sklearn.datasets import load_iris
    iris = load_iris()
    df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                         columns= iris['feature_names'] + ['target'])
    df = df.loc[df['target'] == 1]
```

Plot the histogram

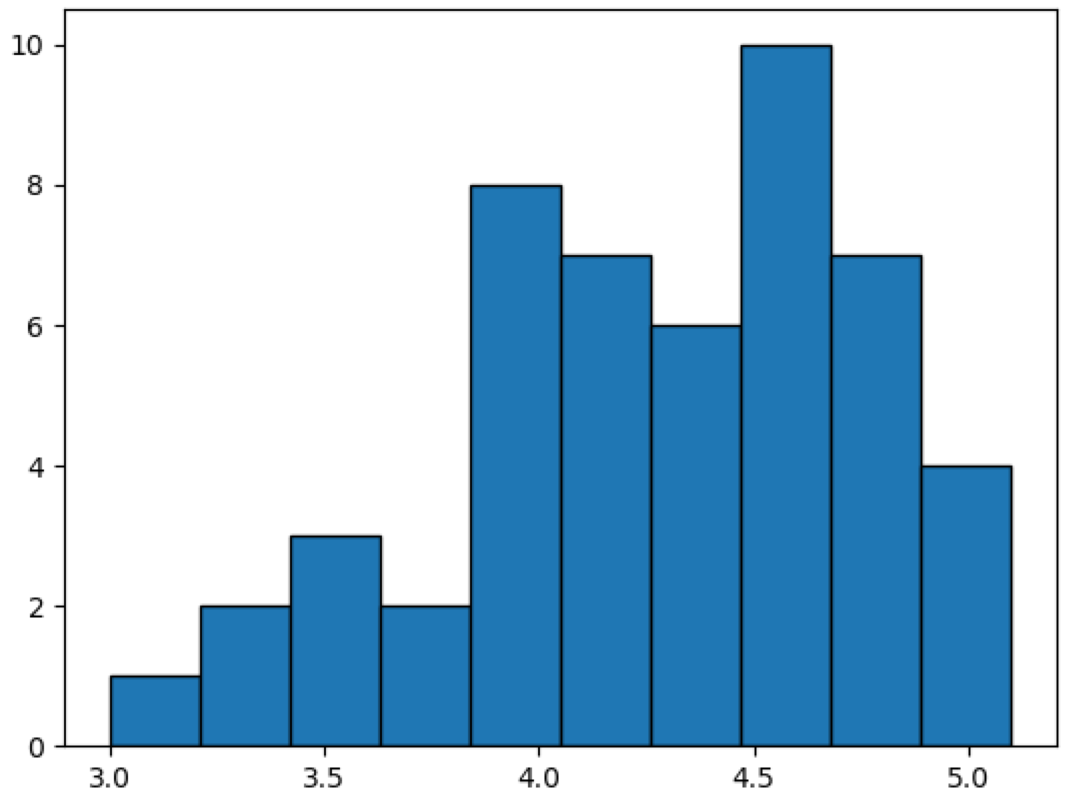
- Plot a histogram of the petal lengths of his 50 samples of *Iris versicolor*.
- Use matplotlib/seaborn's default settings. Recall that to specify the default seaborn style, you can use `sns.set()`; where sns is the alias that seaborn is imported as.
- You view the headers of the dataframe by using `print list(df)`.



Answer

In:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                     columns= iris['feature_names'] + ['target'])
df = df.loc[df['target'] == 1]
_=plt.hist(df['petal length (cm)'], histtype='bar', ec='black')
plt.show()
```

Out:



Practice: axis labels

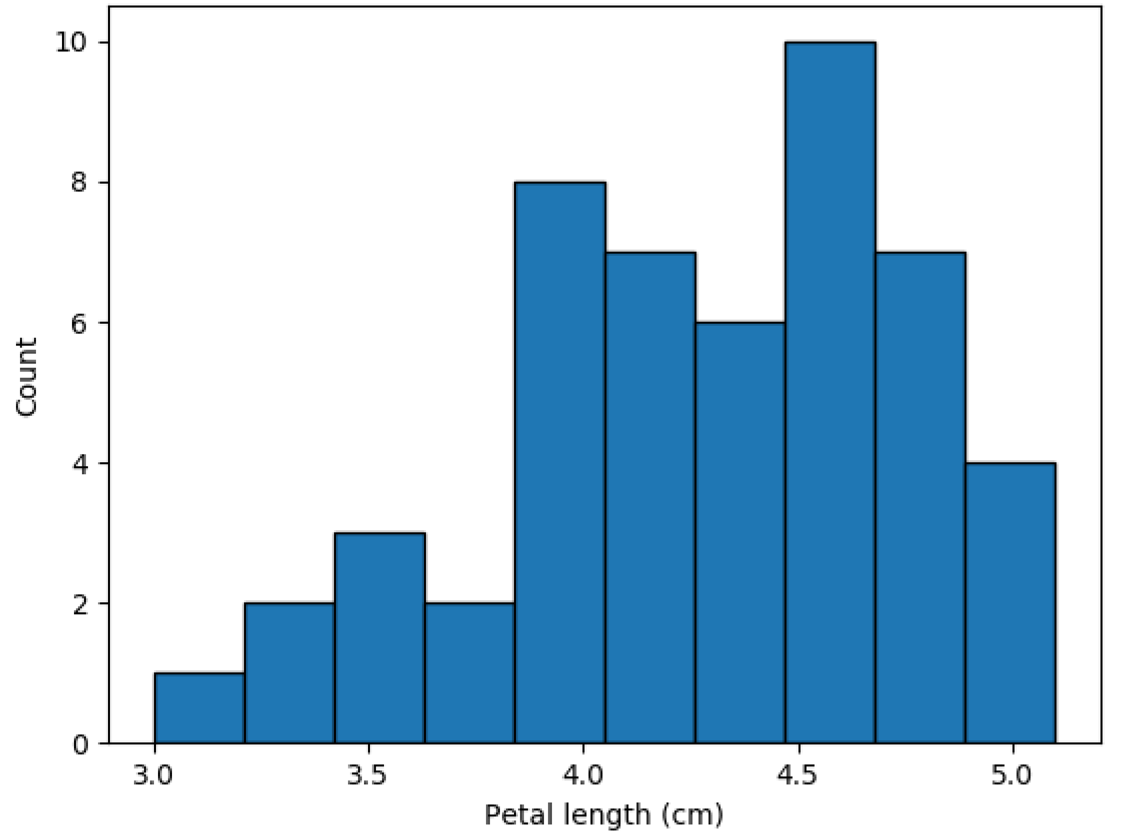
- In the last graph, we commit a cardinal sin of data analysis… We didn’t label our axis.
- Let’s do that now.
- The x-axis should read ‘Petal length (cm)’ and the y-axis should read ‘Count’.
- Assign both axis to the dummy variable of _.

Answer

In:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                     columns= iris['feature_names'] + ['target'])
df = df.loc[df['target'] == 1]
_=plt.hist(df['petal length (cm)'], histtype='bar', ec='black')
_=plt.xlabel('Petal length (cm)')
_=plt.ylabel('Count')
plt.show()
```

Out:



Practice: bins

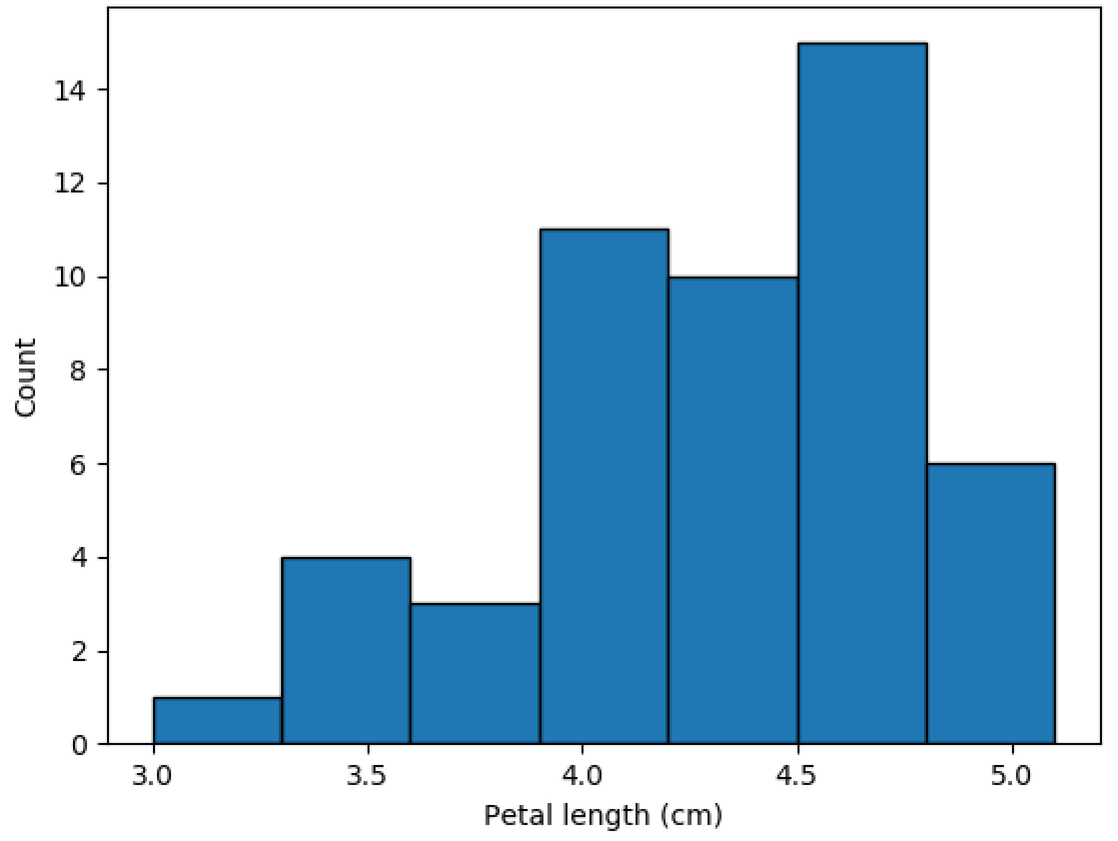
- The histogram you just made had ten bins. This is the default of matplotlib.
- However, the "square root rule" is a commonly-used rule of thumb for choosing number of bins.
- Choose the number of bins to be the square root of the number of samples and plot the histogram again.
- Remember two things:
    1. You specify the number of bins using the `bins` keyword argument of `plt.hist()`
    2. Numpy will probably be able to calculate a square root for you.

Answer

In:

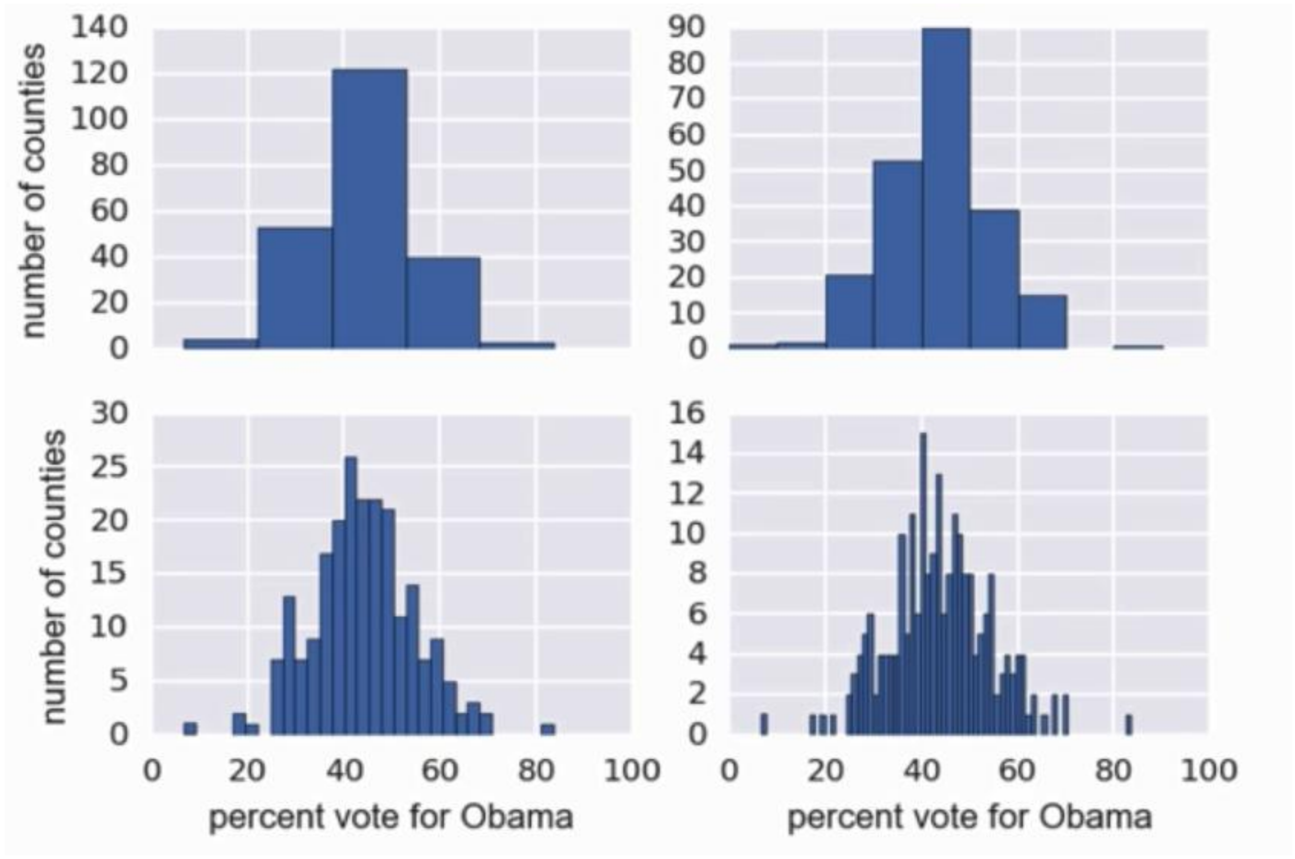
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                     columns= iris['feature_names'] + ['target'])
df = df.loc[df['target'] == 1]

length = len(df)
number_of_Bins = np.sqrt(length)
n_bins = int(7)

_=plt.hist(df['petal length (cm)'], histtype='bar', ec='black', bins=n_bins)
_=plt.xlabel('Petal length (cm)')
_=plt.ylabel('Count')
plt.show()
```

Out:



An issue with histograms

- Histograms have two issues. First, we are not plotting all of the data.
- Second, histograms can plot the same data, and yet, look very different depending on how the bins are set.
- We have encountered the "square root rule" for calculating bins, but bin choice is still very much arbitrary.
- This leads to…

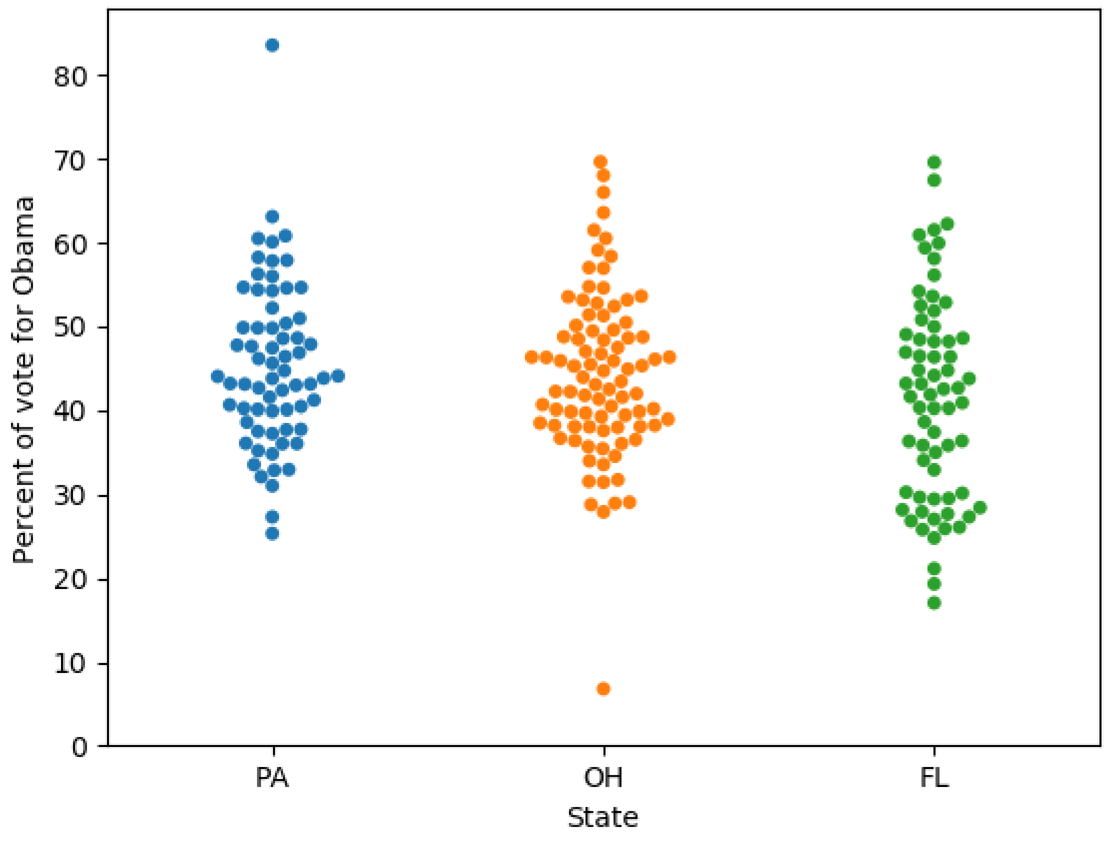


Binning bias

- You may interpret your data different for two different choices of bins.
- To remedy these two issues, we can use a….

Bee swarm plot

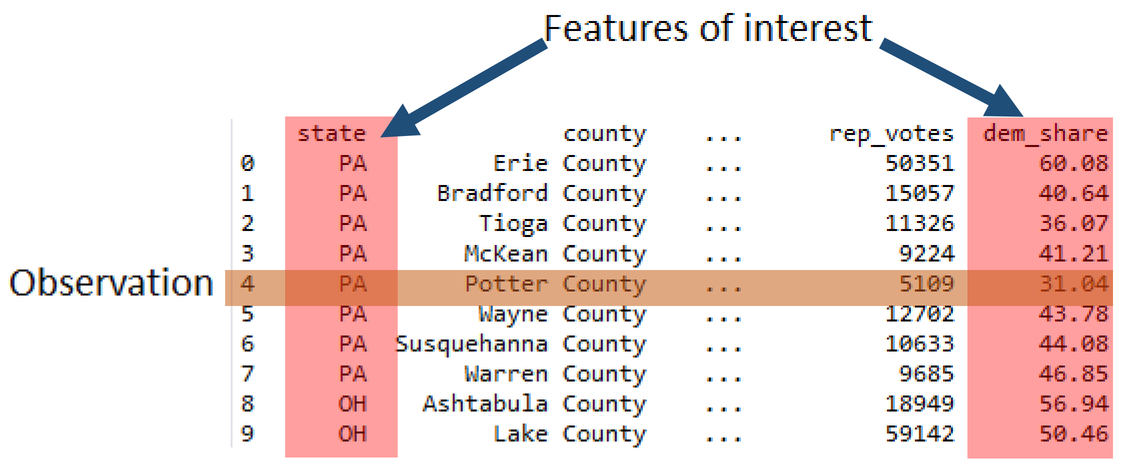
- Here, we see a plot of the vote totals in each of the three swing states.
- Each point in the plot represents the share of the vote that Obama got in a single county.
- The position along the y-axis is the quantitative information.
- The data position along the x-axis is presented the way it is merely to make the graph readable; it does not provide any information.



A requirement for swarm plots

In order to plot one, bee swarm plots require your data to be in a well organised data frame, where:

- Each column is a feature and
- Each row an observation

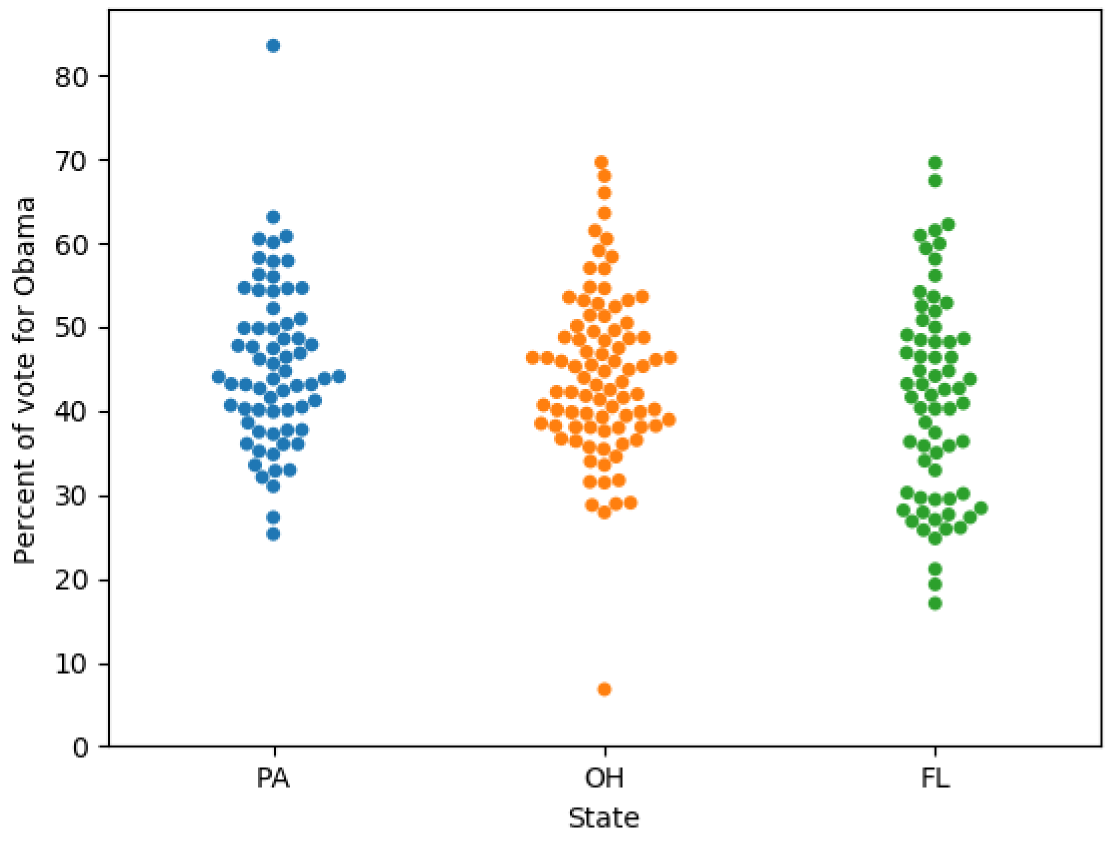

Features of interest
Observation

```
  state            county  ...  rep_votes  dem_share
0    PA       Erie County  ...      50351      60.08
1    PA   Bradford County  ...      15057      40.64
2    PA      Tioga County  ...      11326      36.07
3    PA     McKean County  ...       9224      41.21
4    PA     Potter County  ...       5109      31.04
5    PA      Wayne County  ...      12702      43.78
6    PA Susquehanna County  ...     10633      44.08
7    PA     Warren County  ...       9685      46.85
8    OH  Ashtabula County  ...      18949      56.94
9    OH       Lake County  ...      59142      50.46
```

Plotting a bee swarm plot

In:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
df_swing = pd.read_csv('C:\Users\lb690\Google Drive\Teaching\St
_=sns.swarmplot(x='state', y='dem_share', data=df_swing)
_=plt.xlabel('State')
_=plt.ylabel('Percent of vote for Obama')
plt.show()
```

Out:

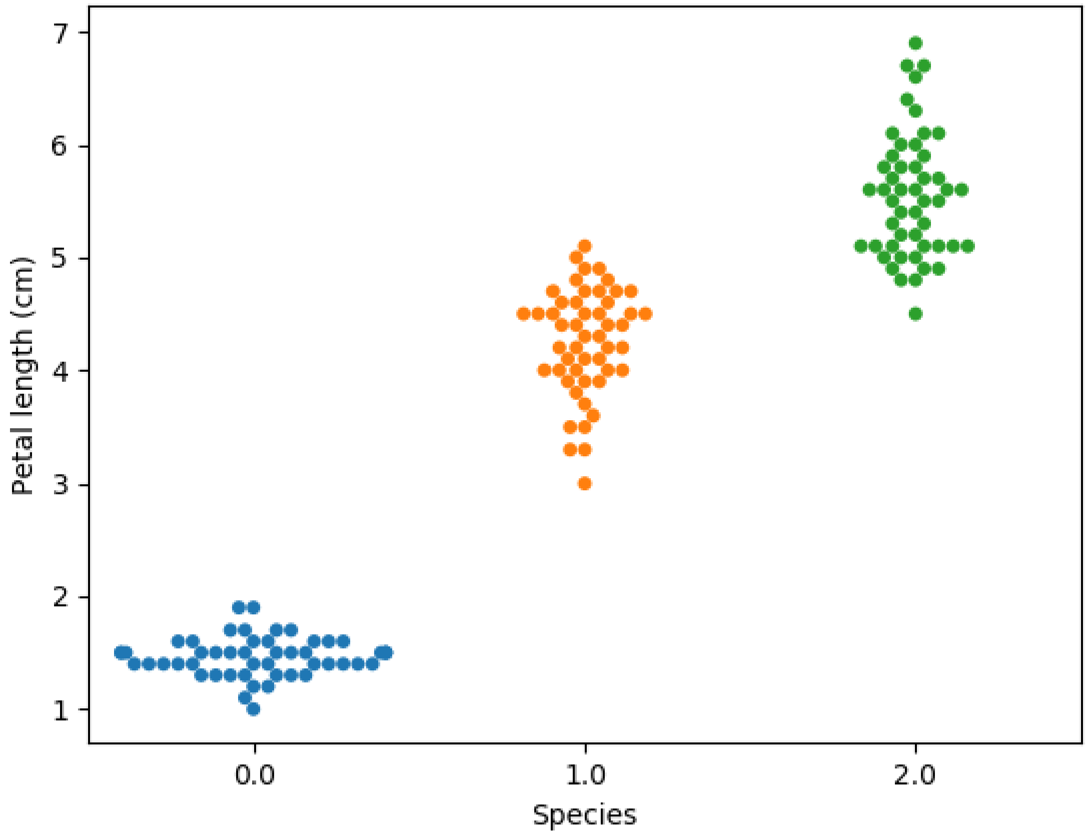


- Again, we can see that Obama got less than 50% of the votes in each of the three swing states.

Practice: Bee swarm plot

- Make a bee swarm plot of the iris petal lengths.
- Your x-axis should contain each of the three species, and the y-axis the petal lengths.
- The data frame lists the species as 0.0, 1.0, and 2.0. This is how they'll appear in your plot. Don't worry about that for the time being.
- Remember to convert the data to a pandas data frame. If you need to do it again, the code is:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
#print iris
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                     columns= iris['feature_names'] + ['target'])
```

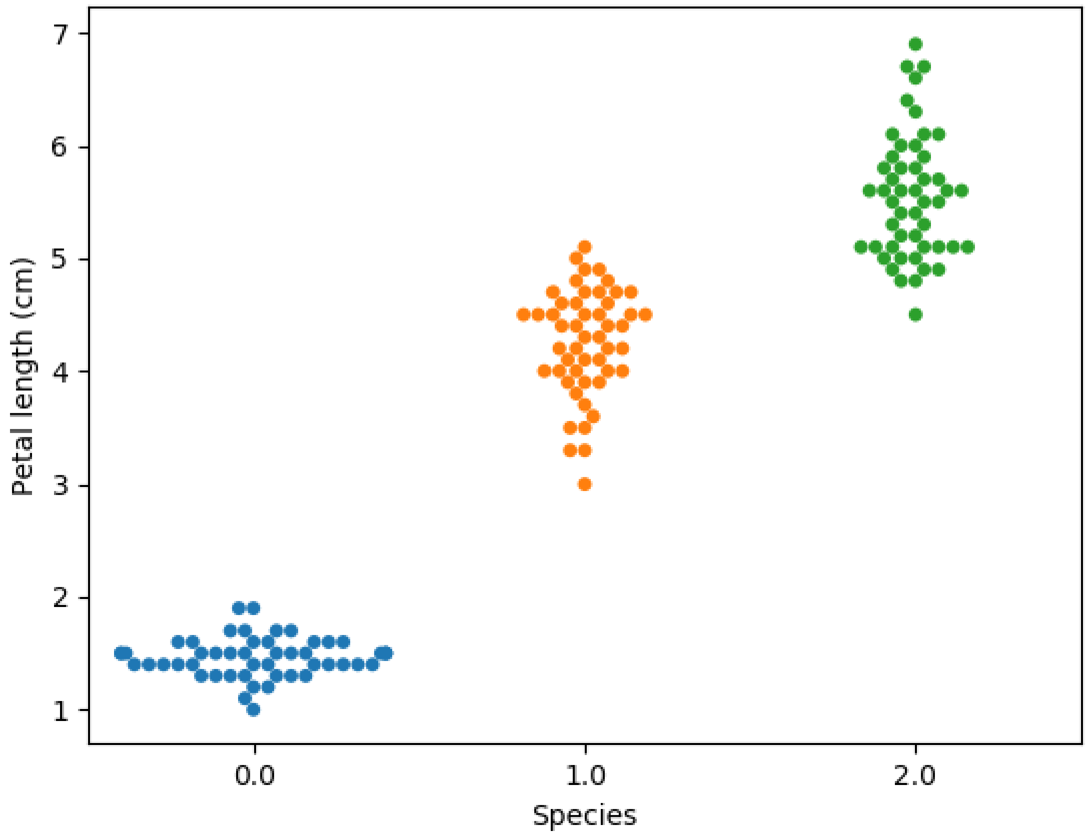


Answer

In:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
iris = load_iris()
#print iris
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                     columns= iris['feature_names'] + ['target'])
_=sns.swarmplot(x='target', y='petal length (cm)', data=df)
_=plt.ylabel('Petal length (cm)')
_=plt.xlabel('Species')
plt.show()
```

Out:

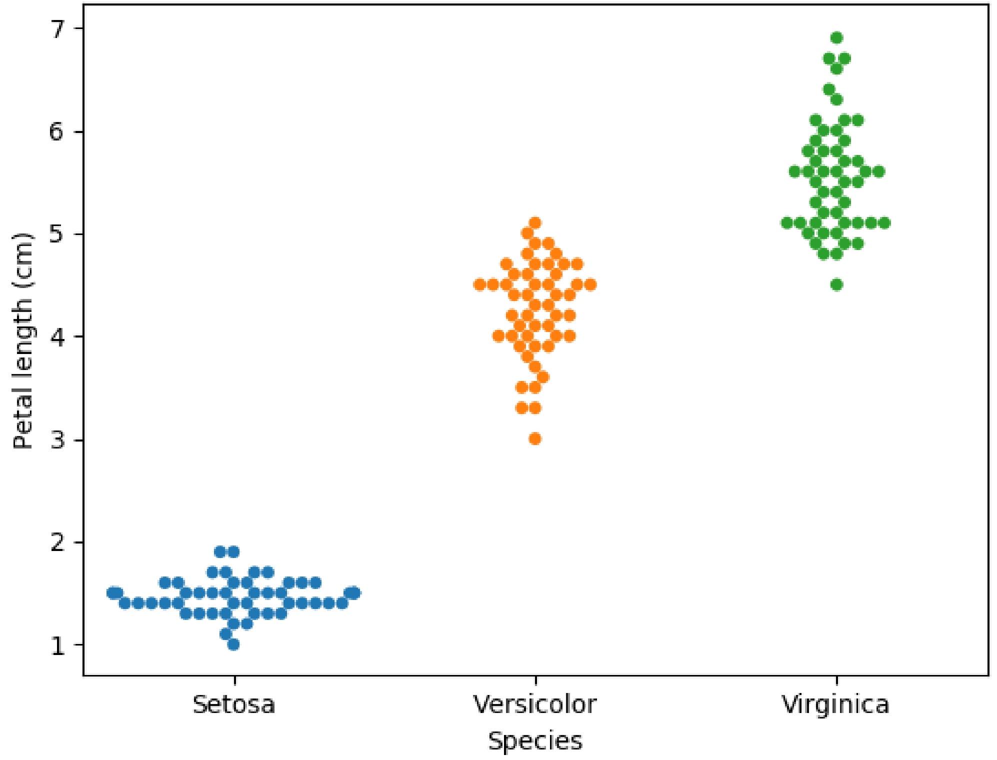


Editing ticks

- ‘Ticks’ refers to the number or word scales on the y-axis and x-axis.
- In the graph we just produce, the x ticks were 0.0, 1.0, and 2.0.
- We can manually alter tick values in Seaborn/matplotlib:

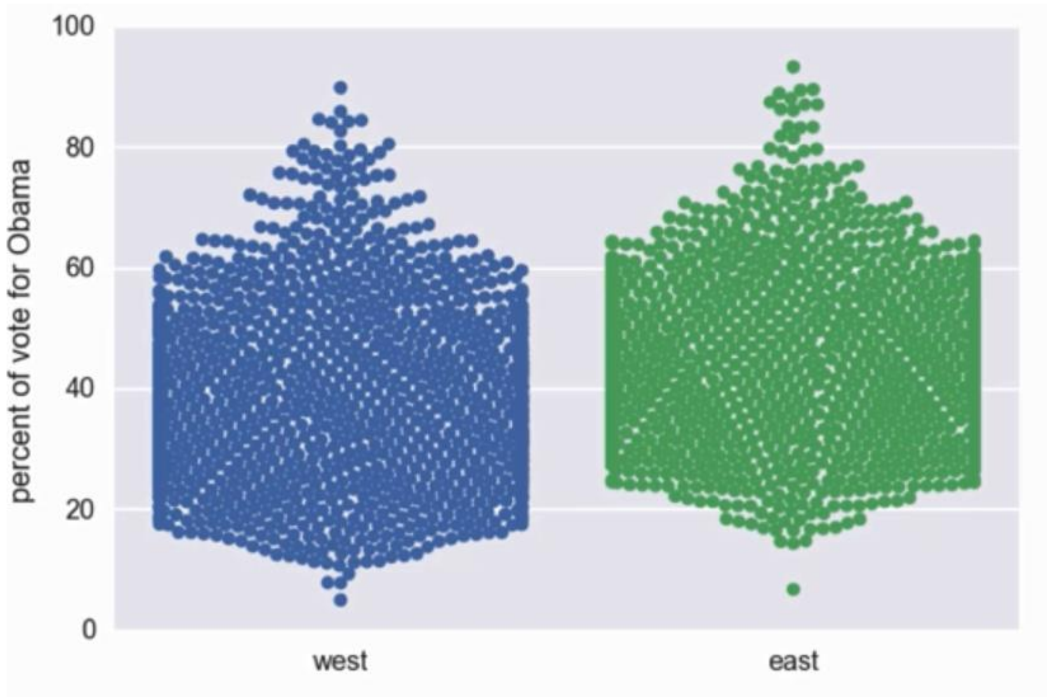
In:
```
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                     columns= iris['feature_names'] + ['target'])
labels = ['Setosa', 'Versicolor', 'Virginica']
_=sns.swarmplot(x='target', y='petal length (cm)', data=df)
_.set_xticklabels(labels)
_=plt.ylabel('Petal length (cm)')
_=plt.xlabel('Species')
plt.show()
```

Out:



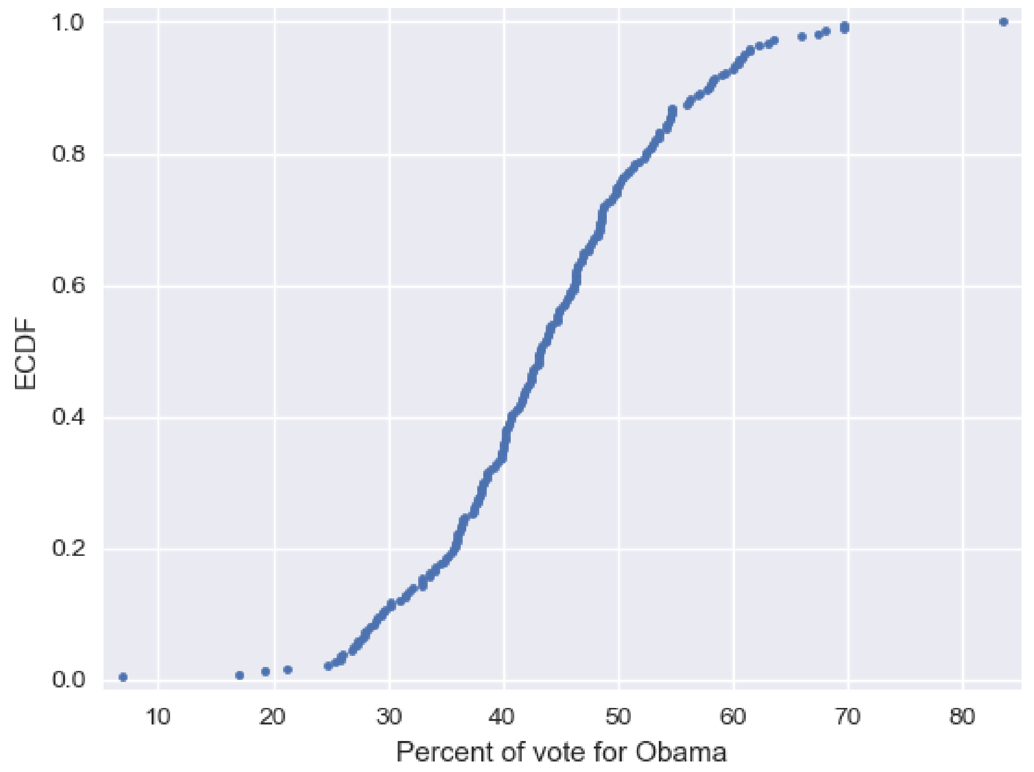
Limits of bee swarm plots

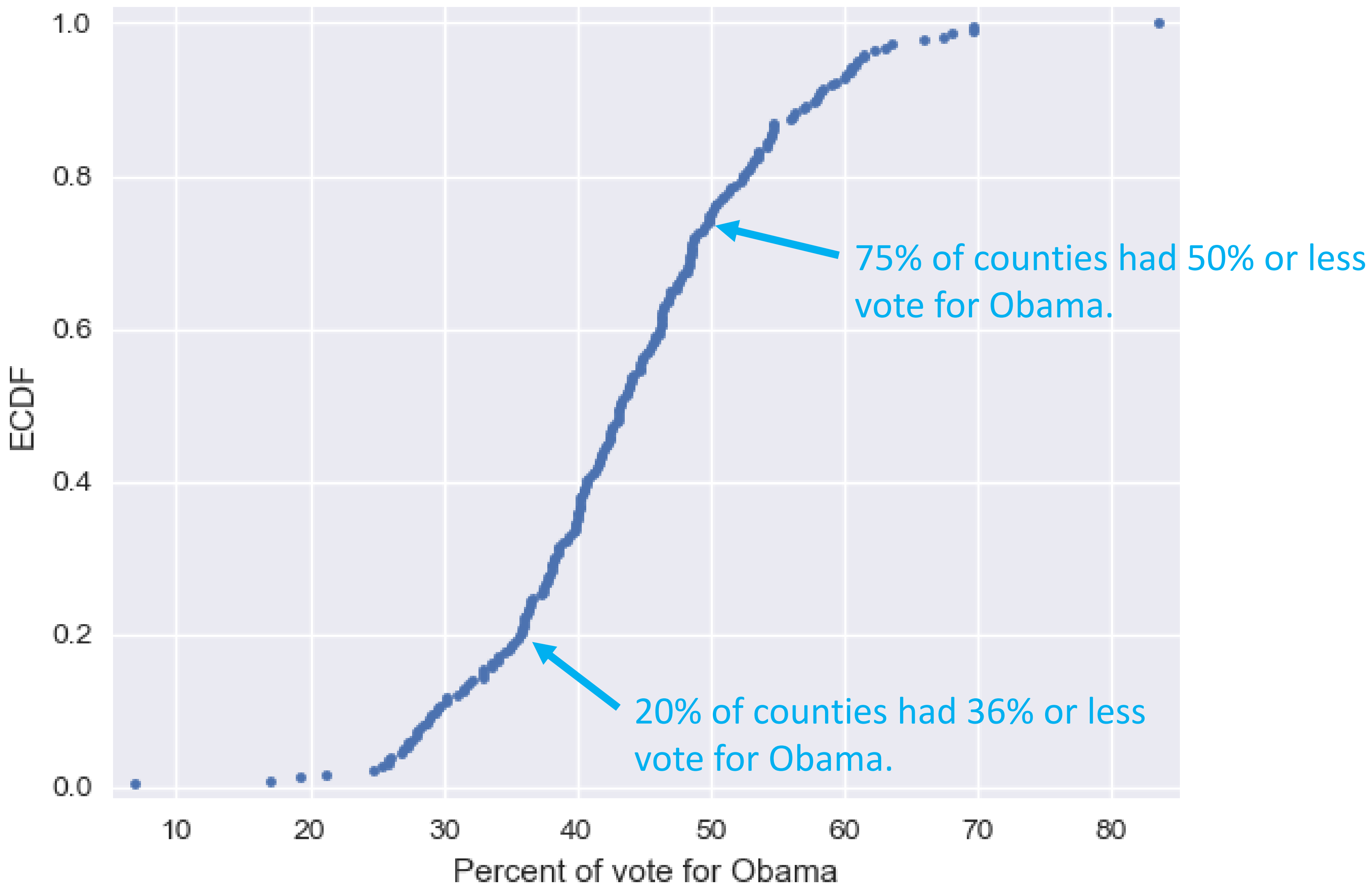
- Bee swarm plots are clearly useful.
- However, they do have their limitations.
- For example, if we were to plot of the county level voting data for all states east of the Mississippi River and all states west.
- We create the same plot as before, but with a data frame that features all states, with each state being classified as east or west of the river.
- The overlapping data points in the resulting graph obscures much of the information that could be gained from the data.



Plotting all data: Empirical cumulative distribution functions (ECDFs)

- As an alternative, we could calculate an ECDF.
- Here we have an ECDF that shows the percentage of swing state votes that went to Obama.
- An *x* value of an ECDF is the quantity you are measure; i.e. percentage of votes.
- The *y* value is the fraction of data points that have a value smaller than the corresponding *x* value. For example…



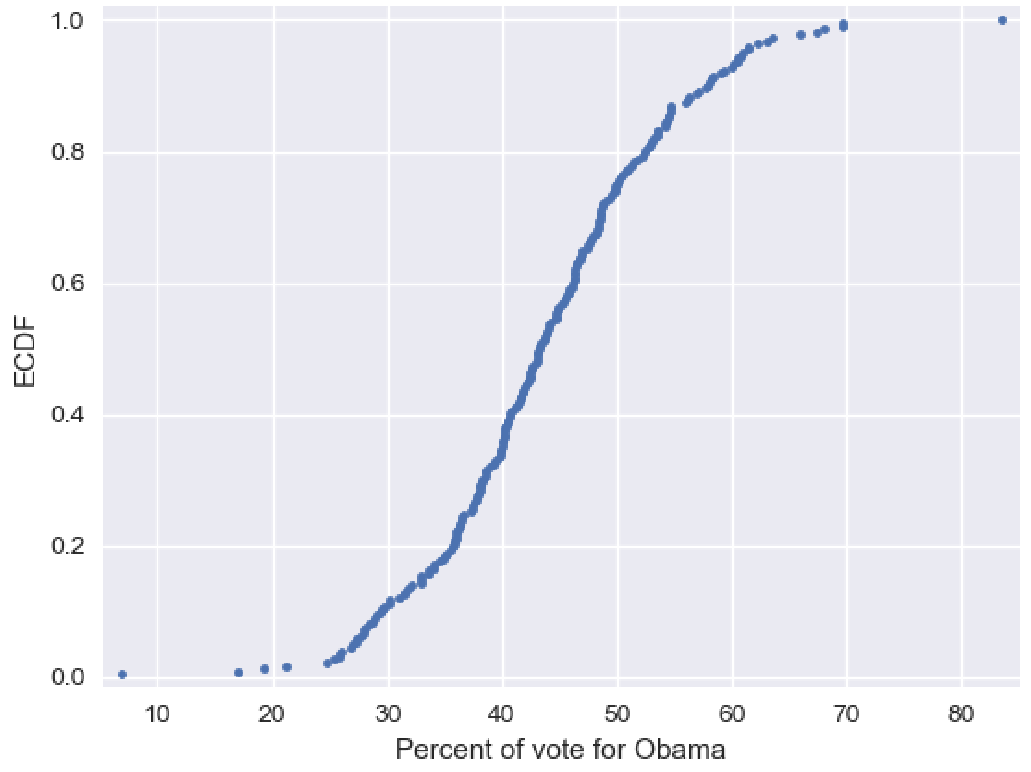

1.0
0.8
0.6
0.4
0.2
0.0
ECDF
10
20
30
40
50
60
70
80
Percent of vote for Obama
75% of counties had 50% or less vote for Obama.
20% of counties had 36% or less vote for Obama.

Plotting an ECDF

In:

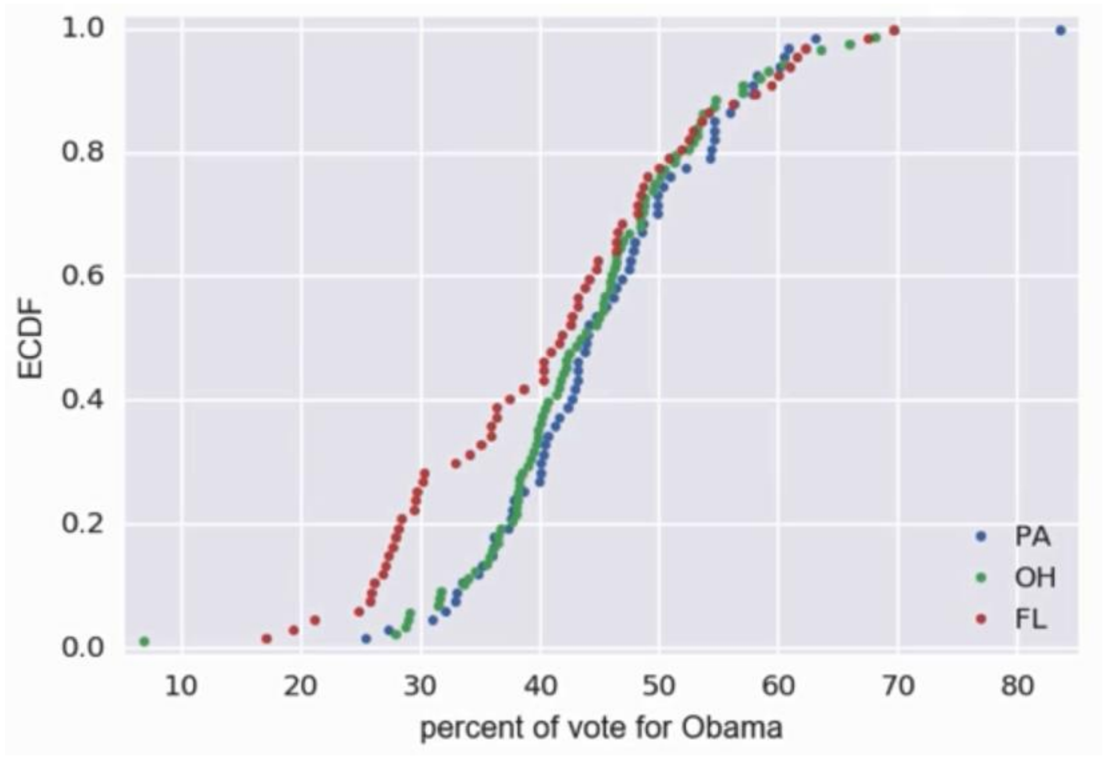
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from decimal import *
sns.set()
df_swing = pd.read_csv('C:\Users\lb690\Google Drive\Teachin
x=np.sort(df_swing['dem_share'])
y=np.arange(1, Decimal(len(x))+1)/Decimal(len(x))
_=plt.plot(x, y, marker='.', linestyle='none')
_=plt.xlabel('Percent of vote for Obama')
_=plt.ylabel('ECDF')
plt.margins(0.02) #Keeps data off plot edges
plt.show()
```

Out:



Note: If using Python 3, you will not need to include the 'Decimal()' around the variables or include 'from decimal import *' in the preamble. This is library was only developed in order to deal with an issue that early version of Python had when dealing with decimals.

- You can also plot multiple ECDFs on the same plot.
- As an example, here with have an ECDF for each of the three swing states.
- We can see here that Florida had a greater number of republican counties.

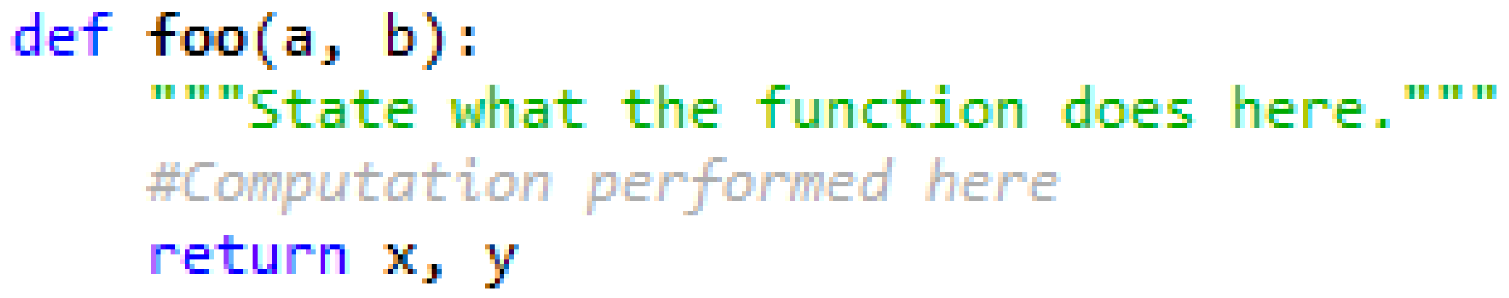


The usefulness of ECDFs

- It often quite useful to plot the ECDF first as part of your workflow.
- This is because it shows all the data and gives a complete picture as to how the data are distributed.

Practice: ECDFs

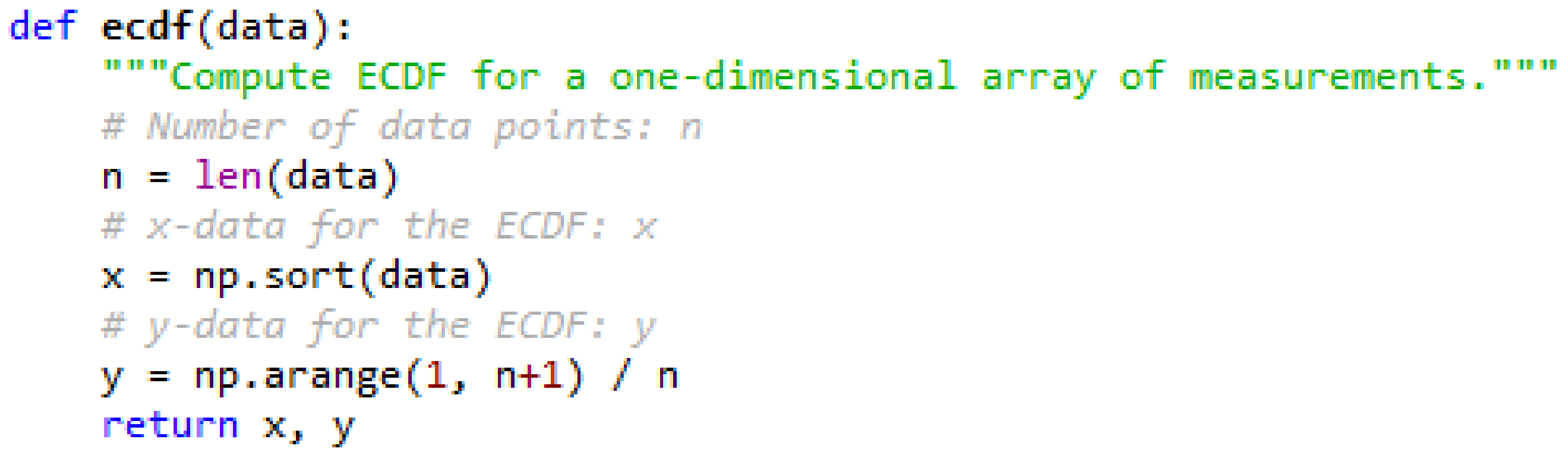
- In this exercise, you will write a function that takes as input a 1D array of data and then returns the *x* and *y* values of the ECDF.
- You will use this function over and over again throughout the remainder of this course. So make sure that you save it a copy.
- You can write your own function, foo(x,y) according to the following skeleton:



```
def foo(a, b):
    """State what the function does here."""
    #Computation performed here
    return x, y
```

- The function foo() takes two arguments *a* and *b* and returns two values *x* and *y*. The function header def foo(a,b): contains the function signature foo(a,b), which consists of the function name, along with its parameters.

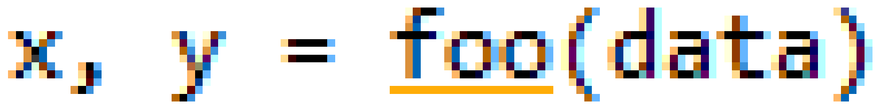
Answer



```python
def ecdf(data):
    """Compute ECDF for a one-dimensional array of measurements."""
    # Number of data points: n
    n = len(data)
    # x-data for the ECDF: x
    x = np.sort(data)
    # y-data for the ECDF: y
    y = np.arange(1, n+1) / n
    return x, y
```

Practice: Plotting an ECDF

- You will now use you `ecdf()` function for the petal lengths of Anderson's Iris versicolor flowers.
- You will then plot the ECDF.
- Recall that your `ecdf()` function returns two arrays so you will need to unpack them. An example of unpacking for the `foo()` function is:



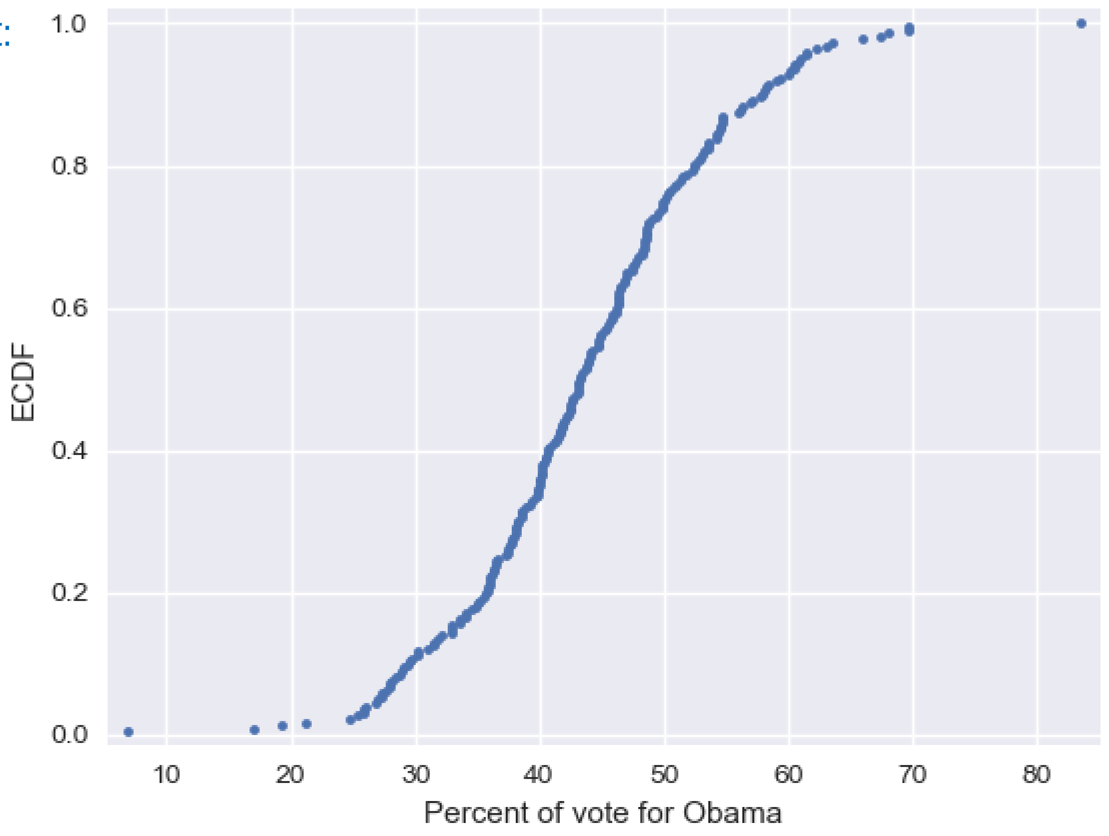
```
x, y = foo(data)
```

Answer

In:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from decimal import *
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                     columns= iris['feature_names'] + ['target'])
df = df.loc[df['target'] == 1]

sns.set()

def ecdf(data):
    """Compute ECDF for a one-dimensional array of measurements."""
    # Number of data points: n
    n = len(data)
    # x-data for the ECDF: x
    x = np.sort(data)
    # y-data for the ECDF: y
    y = np.arange(1, Decimal(n)+1) / Decimal(n)
    return x, y

x, y = ecdf(df['petal length (cm)'])
_=plt.plot(x, y, marker='.', linestyle='none')
_=plt.xlabel('Petal length (cm)')
_=plt.ylabel('ECDF')
plt.margins(0.02) #Keeps data off plot edges
plt.show()
```

Out:



Practice: Comparison of ECDFs

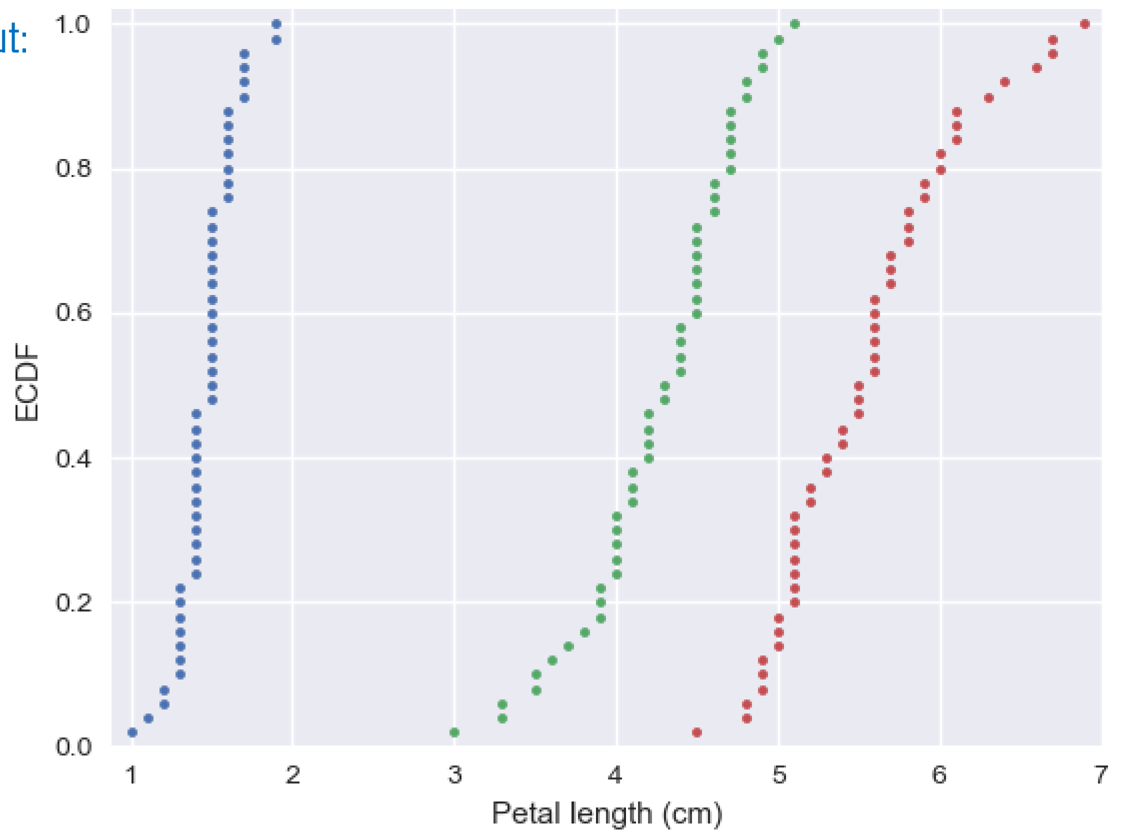
- ECDFs also allow you to compare two or more distributions, although plots do get cluttered if you have too many.
- Here, you will plot ECDFs for the petal lengths of all three iris species.
- You already wrote a function to generate ECDFs so you can put it to good use!
- To overlay all three ECDFs on the same plot, you can use `plt.plot()` three times, once for each ECDF.
- Remember to include `marker='.'` and `linestyle='none'` as arguments inside `plt.plot()`.

Answer

In:

```python
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from decimal import *
from sklearn.datasets import load_iris
iris = load_iris()
df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                     columns= iris['feature_names'] + ['target'])
df_set = df.loc[df['target'] == 0]
df_vers = df.loc[df['target'] == 1]
df_virg = df.loc[df['target'] == 2]

sns.set()

def ecdf(data):
    """Compute ECDF for a one-dimensional array of measurements."""
    # Number of data points: n
    n = len(data)
    # x-data for the ECDF: x
    x = np.sort(data)
    # y-data for the ECDF: y
    y = np.arange(1, Decimal(n)+1) / Decimal(n)
    return x, y

x_set, y_set = ecdf(df_set['petal length (cm)'])
x_vers, y_vers = ecdf(df_vers['petal length (cm)'])
x_virg, y_virg = ecdf(df_virg['petal length (cm)'])
_=plt.plot(x_set, y_set, marker='.', linestyle='none')
_=plt.plot(x_vers, y_vers, marker='.', linestyle='none')
_=plt.plot(x_virg, y_virg, marker='.', linestyle='none')
_=plt.xlabel('Petal length (cm)')
_=plt.ylabel('ECDF')
plt.margins(0.02) #Keeps data off plot edges
plt.show()
```

Out:



Any questions?